

**TUGAS TEORI**  
**SISTEM TERDISTRIBUSI**  
**“TUTORIAL JAVA STREAM”**



**DISUSUN OLEH:**  
**GITHANI RIZKYKA PASYA**  
**2111083009**  
**TRPL 3B**

**DOSEN PENGAMPU :**  
**ERVAN ASRI, S.Kom., M.Kom**

**SEMESTER V**  
**JURUSAN TEKNOLOGI INFORMASI**  
**PROGRAM STUDI D4 TEKNOLOGI REKAYASA PERANGKAT LUNAK**  
**POLITEKNIK NEGERI PADANG**

## Tutorial Java Stream

Java Stream API diperkenalkan pada rilis JDK 8 dengan fungsi utama untuk mempermudah pemrosesan elemen-elemen data secara kolektif menggunakan operasi-operasi fungsional. Java Stream API disediakan pada paket `java.util.stream`. Selain bisa digunakan secara independen, API ini juga bisa diakses melalui Java Collection API.

Dalam *stream* API, terdapat 4 *interface* inti yaitu:

`IntStream`, `LongStream`, `DoubleStream` dan `Stream<T>`. Sesuai namanya, masing-masing *stream* digunakan untuk elemen dengan tipe Integer, Long, dan Double terkecuali *interface* `Stream<T>`.

Ada beberapa operasi pada Java Stream :

### Operasi Menengah

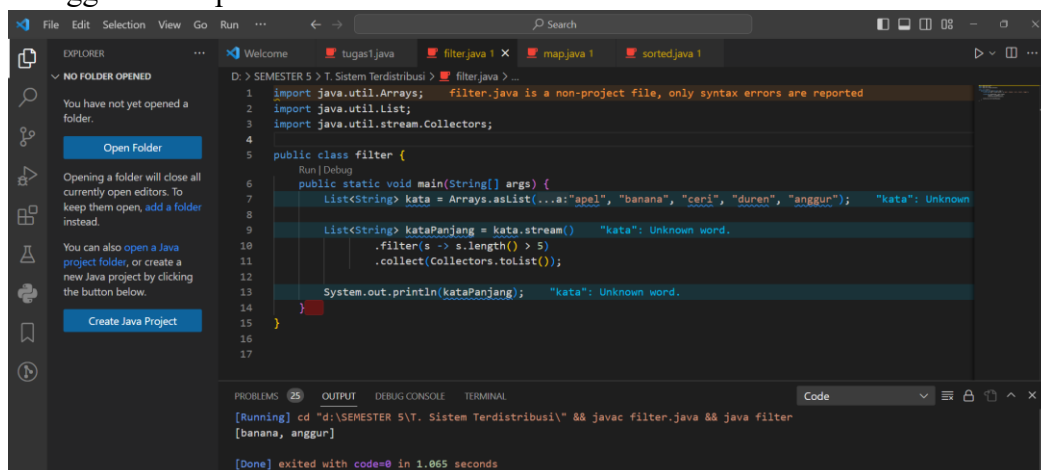
1. `map`: Metode `map` digunakan untuk mengembalikan aliran yang terdiri dari hasil penerapan fungsi yang diberikan ke elemen aliran ini.
2. `filter`: Metode `filter` digunakan untuk memilih elemen sesuai Predikat yang diteruskan sebagai argumen.
3. `sorted`: Metode yang diurutkan digunakan untuk mengurutkan aliran.

### Operasi Terminal

1. `kumpulkan`: Metode pengumpulan digunakan untuk mengembalikan hasil operasi perantara yang dilakukan pada aliran.
2. `forEach`: Metode `forEach` digunakan untuk melakukan iterasi melalui setiap elemen aliran.
3. `pengurangan`: Metode pengurangan digunakan untuk mengurangi elemen aliran menjadi satu nilai. Metode pengurangan menggunakan `BinaryOperator` sebagai parameter.

Dan berikut beberapa Contoh program pada operasi Java Stream :

#### 1. Menggunakan operasi filter

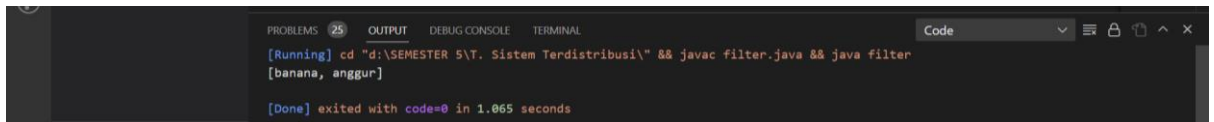


```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 public class filter {
6     public static void main(String[] args) {
7         List<String> kata = Arrays.asList("apel", "banana", "ceri", "duren", "anggur");
8
9         List<String> kataPanjang = kata.stream()
10             .filter(s -> s.length() > 5)
11             .collect(Collectors.toList());
12
13         System.out.println(kataPanjang);
14     }
15 }
```

OUTPUT

```
[Running] cd "d:\SEMESTER 5\T. Sistem Terdistribusi\" && javac filter.java && java filter
[banana, anggur]
[Done] exited with code=0 in 1.065 seconds
```

Output :



Penjelasan :

Kode di atas membuat sebuah list `kata` yang berisi beberapa kata.

Kemudian, kita menggunakan Stream pada list `kata` dengan `kata.stream()`.

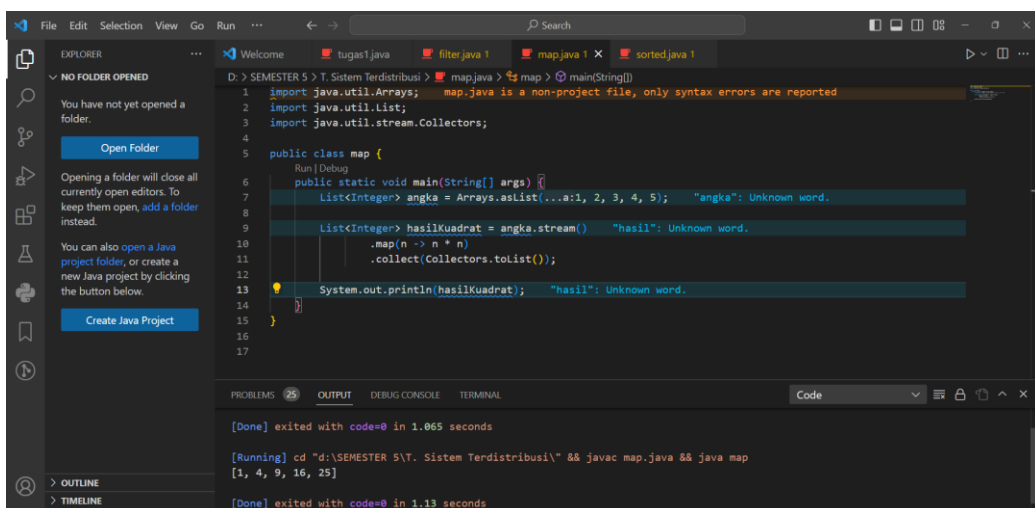
Pada Stream tersebut, kita menggunakan operasi `filter`. `.filter()` digunakan untuk menyaring elemen-elemen yang memenuhi suatu kondisi tertentu. Dalam contoh ini, kondisinya adalah panjang string lebih dari 5 karakter, yang diwakili oleh ekspresi lambda `s -> s.length() > 5`.

Hasil dari operasi `filter` ini adalah sebuah Stream baru yang hanya berisi kata-kata yang panjangnya lebih dari 5 huruf.

Akhirnya, kita menggunakan `.collect(Collectors.toList())` untuk mengumpulkan hasilnya ke dalam List baru `kataPanjang`.

Hasilnya adalah List baru yang hanya berisi kata "banana" karena kata ini adalah satu-satunya yang memiliki panjang lebih dari 5 huruf.

## 2. Menggunakan Operasi Map



Output :



Penjelasan :

Kode di atas membuat sebuah list `angka` yang berisi beberapa angka.

Kemudian, kita menggunakan Stream pada list `angka` dengan `angka.stream()`.

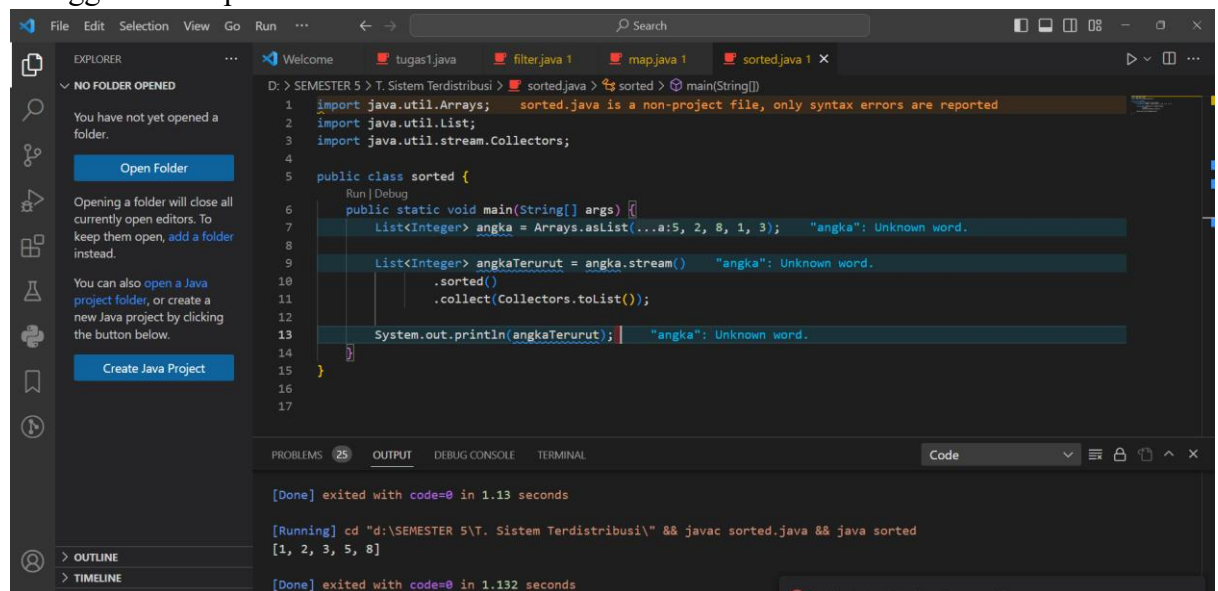
Pada Stream tersebut, kita menggunakan operasi `map`. `.map()` digunakan untuk mengubah setiap elemen dalam Stream sesuai dengan ekspresi lambda yang diberikan. Dalam contoh ini, kita mengkuadratkan setiap angka dengan ekspresi `n -> n * n`.

Hasil dari operasi `map` ini adalah Stream baru yang berisi hasil pengkuadratan setiap angka dalam list.

Akhirnya, kita menggunakan `.collect(Collectors.toList())` untuk mengumpulkan hasilnya ke dalam List baru `hasilKuadrat`.

Hasilnya adalah List baru yang berisi hasil kuadrat dari setiap angka dalam list asalnya.

### 3. Menggunakan Operasi Sorted



The screenshot shows an IDE with a Java file named `sorted.java`. The code imports `java.util.Arrays`, `java.util.List`, and `java.util.stream.Collectors`. It defines a `sorted` class with a `main` method. In the `main` method, a `List<Integer>` named `angka` is created with the values `5, 2, 8, 1, 3`. Then, a new `List<Integer>` named `angkaTerurut` is created by calling `angka.stream().sorted().collect(Collectors.toList())`. Finally, `System.out.println(angkaTerurut)` is called to print the sorted list. The output window shows the command `cd "d:\SEMESTER 5\T. Sistem Terdistribusi\" && javac sorted.java && java sorted` and the output `[1, 2, 3, 5, 8]`.

```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 public class sorted {
6     public static void main(String[] args) {
7         List<Integer> angka = Arrays.asList(5, 2, 8, 1, 3);
8
9         List<Integer> angkaTerurut = angka.stream()
10             .sorted()
11             .collect(Collectors.toList());
12
13         System.out.println(angkaTerurut);
14     }
15 }
16
17
```

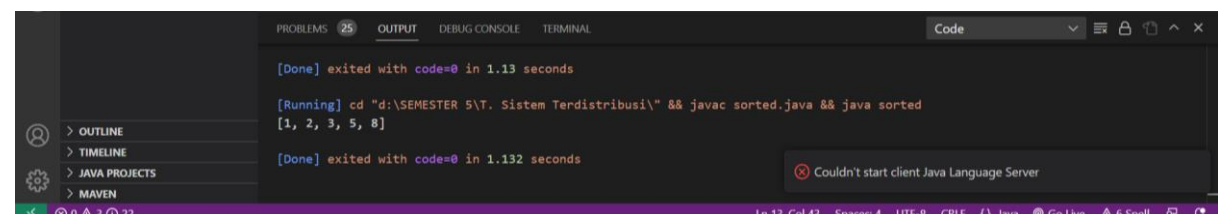
[Done] exited with code=0 in 1.13 seconds

[Running] cd "d:\SEMESTER 5\T. Sistem Terdistribusi\" && javac sorted.java && java sorted

[1, 2, 3, 5, 8]

[Done] exited with code=0 in 1.132 seconds

Output:



The screenshot shows the same IDE as before, but with the output window open. The output shows the command `cd "d:\SEMESTER 5\T. Sistem Terdistribusi\" && javac sorted.java && java sorted` and the output `[1, 2, 3, 5, 8]`. There is also a message in the bottom right corner that says "Couldn't start client Java Language Server".

[Done] exited with code=0 in 1.13 seconds

[Running] cd "d:\SEMESTER 5\T. Sistem Terdistribusi\" && javac sorted.java && java sorted

[1, 2, 3, 5, 8]

[Done] exited with code=0 in 1.132 seconds

Couldn't start client Java Language Server

Penjelasan :

Kode di atas membuat sebuah list `angka` yang berisi beberapa angka yang tidak terurut.

Kemudian, kita menggunakan Stream pada list `angka` dengan `angka.stream()`.

Pada Stream tersebut, kita menggunakan operasi `sorted`. `.sorted()` digunakan untuk mengurutkan elemen-elemen dalam Stream. Dalam contoh ini, kita tidak memberikan kriteria pengurutan khusus, sehingga angka-angka akan diurutkan secara ascending (menaik) secara default.

Hasil dari operasi `sorted` ini adalah Stream baru yang berisi angka-angka yang sudah terurut.

Akhirnya, kita menggunakan `.collect(Collectors.toList())` untuk mengumpulkan hasilnya ke dalam List baru `angkaTerurut`.

Hasilnya adalah List baru yang berisi angka-angka yang sudah diurutkan dari terkecil ke terbesar.